

# Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/FR05/000073

International filing date: 13 January 2005 (13.01.2005)

Document type: Certified copy of priority document

Document details: Country/Office: FR  
Number: 0400291  
Filing date: 14 January 2004 (14.01.2004)

Date of receipt at the International Bureau: 30 March 2005 (30.03.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)



World Intellectual Property Organization (WIPO) - Geneva, Switzerland  
Organisation Mondiale de la Propriété Intellectuelle (OMPI) - Genève, Suisse



PCT/FR 2005 / 000073

# BREVET D'INVENTION

CERTIFICAT D'UTILITÉ - CERTIFICAT D'ADDITION

## COPIE OFFICIELLE

Le Directeur général de l'Institut national de la propriété industrielle certifie que le document ci-annexé est la copie certifiée conforme d'une demande de titre de propriété industrielle déposée à l'Institut.

Fait à Paris, le 06 DEC. 2004

Pour le Directeur général de l'Institut  
national de la propriété industrielle  
Le Chef du Département des brevets

Martine PLANCHE

INSTITUT  
NATIONAL DE  
LA PROPRIÉTÉ  
INDUSTRIELLE

SIEGE  
26 bis, rue de Saint-Petersbourg  
75800 PARIS cedex 08  
Téléphone : 33 (0)1 53 04 53 04  
Télécopie : 33 (0)1 53 04 45 23  
www.inpi.fr

PUBLI-GRAM

ETABLISSEMENT PUBLIC NATIONAL

CRÉÉ PAR LA LOI N° 51-444 DU 19 AVRIL 1951



INSTITUT  
NATIONAL DE  
LA PROPRIÉTÉ  
INDUSTRIELLE  
26 bis, rue de Saint Pétersbourg  
75800 Paris Cedex 08  
Téléphone : 33 (1) 53 04 53 04 Télécopie : 33 (1) 42 94 86 54

1er dépôt

# BREVET D'INVENTION CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI



N° 11354\*03.

## REQUÊTE EN DÉLIVRANCE

page 1/2



Cet imprimé est à remplir lisiblement à l'encre noire

DB 540 W / 210502

<b>REMISE DES PIÈCES</b> DATE <b>14 JAN 2004</b> LIEU <b>75 INPI PARIS 34 SP</b> N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI <b>0400291</b> DATE DE DÉPÔT ATTRIBUÉE PAR L'INPI <b>14 JAN. 2004</b>		<b>1 NOM ET ADRESSE DU DEMANDEUR OU DU MANDATAIRE</b> À QUI LA CORRESPONDANCE DOIT ÊTRE ADRESSÉE  CABINET BEAU DE LOMENIE 158, rue de l'Université 75340 PARIS CEDEX 07 FRANCE	
<b>Vos références pour ce dossier</b> (facultatif) 1H257420/18.JBT			
<b>Confirmation d'un dépôt par télécopie</b>		<input type="checkbox"/> N° attribué par l'INPI à la télécopie	
<b>2 NATURE DE LA DEMANDE</b>		<b>Cochez l'une des 4 cases suivantes</b>	
Demande de brevet	<input checked="" type="checkbox"/>		
Demande de certificat d'utilité	<input type="checkbox"/>		
Demande divisionnaire	<input type="checkbox"/>		
<i>Demande de brevet initiale</i>	N°	Date	
<i>ou demande de certificat d'utilité initiale</i>	N°	Date	
Transformation d'une demande de brevet européen <i>Demande de brevet initiale</i>	<input type="checkbox"/>	N°	Date
<b>3 TITRE DE L'INVENTION</b> (200 caractères ou espaces maximum)  Système de génération automatique de codes optimisés			
<b>4 DÉCLARATION DE PRIORITÉ</b> <b>OU REQUÊTE DU BÉNÉFICE DE</b> <b>LA DATE DE DÉPÔT D'UNE</b> <b>DEMANDE ANTÉRIEURE FRANÇAISE</b>		Pays ou organisation Date N° Pays ou organisation Date N° Pays ou organisation Date N° <input type="checkbox"/> S'il y a d'autres priorités, cochez la case et utilisez l'imprimé «Suite»	
<b>5 DEMANDEUR</b> (Cochez l'une des 2 cases)		<input checked="" type="checkbox"/> Personne morale <input type="checkbox"/> Personne physique	
Nom ou dénomination sociale		COMMISSARIAT A L'ENERGIE ATOMIQUE	
Prénoms			
Forme juridique		Etablissement Public de caractère scientifique, technique et industriel	
N° SIREN			
Code APE-NAF			
Domicile ou siège	Rue	31-33, rue de la Fédération	
	Code postal et ville	75 015 PARIS	
	Pays	FRANCE	
Nationalité		Française	
N° de téléphone (facultatif)		N° de télécopie (facultatif)	
Adresse électronique (facultatif)			
<input checked="" type="checkbox"/> S'il y a plus d'un demandeur, cochez la case et utilisez l'imprimé «Suite»			

Remplir impérativement la 2<sup>ème</sup> page

REMISE DES PIÈCES DATE <b>14 JAN 2004</b> LIEU <b>75 INPI PARIS 34 SP</b> N° D'ENREGISTREMENT <b>0400291</b> NATIONAL ATTRIBUÉ PAR L'INPI		Réservé à l'INPI	1H257420/18.JBT	DB 540 W / 210502
<b>6 MANDATAIRE (s'il y a lieu)</b> Nom Prénom Cabinet ou Société <b>CABINET BEAU DE LOMENTIE</b> N° de pouvoir permanent et/ou de lien contractuel Adresse Rue <b>158, rue de l'Université</b> Code postal et ville <b>75 340 PARIS CEDEX 07</b> Pays <b>FRANCE</b> N° de téléphone (facultatif) <b>01.44.18.89.00</b> N° de télécopie (facultatif) <b>01.44.18.04.23</b> Adresse électronique (facultatif)				
<b>7 INVENTEUR(S)</b> Les inventeurs sont nécessairement des personnes physiques Les demandeurs et les inventeurs sont les mêmes personnes <input type="checkbox"/> Oui <input checked="" type="checkbox"/> Non : Dans ce cas remplir le formulaire de Désignation d'inventeur(s)				
<b>8 RAPPORT DE RECHERCHE</b> Établissement immédiat ou établissement différé <input checked="" type="checkbox"/> Établissement immédiat <input type="checkbox"/> Établissement différé Paiement échelonné de la redevance (en deux versements) <input type="checkbox"/> Oui <input type="checkbox"/> Non		Uniquement pour une demande de brevet (y compris division et transformation) Uniquement pour les personnes physiques effectuant elles-mêmes leur propre dépôt <input type="checkbox"/> Oui <input type="checkbox"/> Non		
<b>9 RÉDUCTION DU TAUX DES REDEVANCES</b> Réduction pour les personnes physiques <input type="checkbox"/> Requête pour la première fois pour cette invention (joindre un avis de non-imposition) <input type="checkbox"/> Obtenue antérieurement à ce dépôt pour cette invention (joindre une copie de la décision d'admission à l'assistance gratuite ou indiquer sa référence) : AG				
<b>10 SÉQUENCES DE NUCLEOTIDES ET/OU D'ACIDES AMINÉS</b> Le support électronique de données est joint <input type="checkbox"/> Oui <input type="checkbox"/> Non La déclaration de conformité de la liste de séquences sur support papier avec le support électronique de données est jointe <input type="checkbox"/> Oui <input type="checkbox"/> Non		<input type="checkbox"/> Cochez la case si la description contient une liste de séquences		
Si vous avez utilisé l'imprimé « Suften », indiquez le numéro de pages jointes				

CHANCELLERIE DU DÉPARTEMENT  
100, rue de la République  
92000 NANTERRE  
ou au service de brevets.

1000-1000-1000  
1000-1000-1000  
1000-1000-1000

VISA DE LA PRÉFECTURE  
1000-1000-1000



26 bis, rue de Saint Pétersbourg  
75800 Paris Cedex 08  
Téléphone : 33 (1) 53 04 53 04 Télécopie : 33 (1) 42 94 86 54

# BREVET D'INVENTION CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI



## REQUÊTE EN DÉLIVRANCE

Page suite N° .../2...



REMISE DES PIÈCES DATE LIEU <b>14 JAN 2004</b> <b>75 INPI PARIS 34 SP</b> N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI <b>0400291</b>		Réservé à l'INPI Cet imprimé est à remplir lisiblement à l'encre noire	DB 829 W / 140301
<b>Vos références pour ce dossier (facultatif)</b>		<b>1H257420.18/JBT</b>	
<b>4 DÉCLARATION DE PRIORITÉ OU REQUÊTE DU BÉNÉFICE DE LA DATE DE DÉPÔT D'UNE DEMANDE ANTÉRIEURE FRANÇAISE</b>		Pays ou organisation Date <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> N° Pays ou organisation Date <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> N° Pays ou organisation Date <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> N°	
<b>5 DEMANDEUR</b>		<b>5 DEMANDEUR</b>	
Nom ou dénomination sociale		CAPS ENTREPRISE	
Prénoms			
Forme juridique		Société par action simplifiée	
N° SIREN		<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	
Code APE-NAF		<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	
Adresse	Rue	Immeuble Le Gallium 80, avenue des Buttes de Coësmes	
	Code postal et ville	35 700 RENNES	
	Pays	FRANCE	
Nationalité		Française	
N° de téléphone (facultatif)			
N° de télécopie (facultatif)			
Adresse électronique (facultatif)			
<b>5 DEMANDEUR</b>		<b>5 DEMANDEUR</b>	
Nom ou dénomination sociale		ASSOCIATION TRANSVERSA	
Prénoms			
Forme juridique		Association loi 1901	
N° SIREN		<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	
Code APE-NAF		<input type="text"/> <input type="text"/> <input type="text"/> <input type="text"/>	
Adresse	Rue	45, avenue des Etats-Unis	
	Code postal et ville	78 000 VERSAILLES	
	Pays	FRANCE	
Nationalité		Française	
N° de téléphone (facultatif)			
N° de télécopie (facultatif)			
Adresse électronique (facultatif)			
<b>10 SIGNATURE DU DEMANDEUR OU DU MANDATAIRE (Nom et qualité du signataire)</b>		<b>VISA DE LA PRÉFECTURE OU DE L'INPI</b>	
Jean-Bruno THEVENET CPI N° 92-1236 Paris, le 14 Janvier 2004			

La loi n°78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés s'applique aux réponses faites à ce formulaire.  
Elle garantit un droit d'accès et de rectification pour les données vous concernant auprès de l'INPI

La présente invention a pour objet un système de génération  
5 automatique de codes optimisés opérationnels sur une plate-forme  
matérielle prédéfinie comportant au moins un processeur, pour un  
domaine d'application prédéterminé, à partir de codes sources soumis par  
des utilisateurs (ces utilisateurs étant compris ici au sens large et incluant  
10 des utilisateurs finaux, mais également des programmeurs d'applications  
ou des programmeurs système).

Depuis l'origine de l'informatique, de nombreux travaux ont été  
réalisés concernant les compilateurs.

Le principe d'un compilateur est d'analyser un code source décrit  
dans un langage de haut niveau puis de générer son équivalent en code  
15 binaire pour la machine cible. En général ce processus est effectué  
statiquement avant l'exécution. Il existe aussi des interpréteurs mettant en  
œuvre des technologies de compilation dynamique qui ont permis de lever  
cette dernière contrainte en offrant la possibilité de générer le code au  
dernier moment lors de l'exécution.

20 Le compilateur est un élément de la chaîne de préparation des  
programmes. Le résultat de la compilation peut être complété par des  
procédures déjà compilées et générées (compilation séparée ou  
bibliothèques), liées statiquement au chargement ou dynamiquement à  
l'exécution.

25 Les compilateurs sont en général organisés en trois phases :

1. Génération de code intermédiaire : à partir du code source, le  
compilateur reconnaît les idiomes et génère une forme abstraite  
indépendante du langage source, communément appelée langage  
intermédiaire. Ce langage est indépendant de la machine cible.
- 30 2. Optimisation haut niveau : dans cette phase sont regroupées un  
certain nombre d'optimisations qui sont en général indépendantes de  
l'architecture cible : propagation de constantes, réduction de force,  
expressions communes... Ces optimisations correspondent en général  
à des techniques classiques connues de nombreux programmeurs.

35

3. Génération de code et optimisation bas niveau ; dans cette phase sont réalisées toutes les opérations et optimisations propres à la machine cible : génération et sélection des instructions, allocation de registres, ordonnancement des instructions, etc.

5 Les qualités d'un compilateur sont non seulement liées à l'architecture cible (performance du code généré) mais aussi au langage source (plus ou moins facile à compiler) et à ses caractéristiques en tant que logiciel (robustesse, richesse des options, vitesse d'exécution, etc.).

10 Sauf le cas particulier de la compilation dynamique, l'ensemble des trois phases ci-dessus doivent être réalisées avant l'exécution et ce dans un temps raisonnable, ce qui limite d'autant la complexité des algorithmes d'optimisation qui peuvent être implémentés dans un compilateur.

15 Une grande part de la recherche sur les compilateurs est donc en premier lieu liée au choix et à la définition du langage source de haut niveau.

20 Les évolutions des compilateurs sont également liées aux évolutions de l'architecture des processeurs et plus précisément aux modèles de performance décrivant le comportement de ces architectures. Comme dans tout problème d'optimisation, une difficulté importante est la définition d'une fonction coût qui soit représentative du temps d'exécution.

25 Les premiers jeux d'instructions avaient des comportements très simples : le temps d'exécution d'une séquence d'instructions s'obtenait tout simplement comme la somme des temps d'exécution de chacune des instructions de la séquence. En conséquence, le processus d'optimisation était très simple et la principale stratégie d'optimisation consistait à réduire le nombre et la complexité des instructions générées.

30 L'apparition des premiers jeux d'instructions CISC ("Complex Instruction Set Computer") a légèrement modifié la donne dans la mesure où des instructions très complexes devenaient disponibles. Le problème d'optimisation devenait alors essentiellement un problème de reconnaissance d'idiomes (« pattern matching »). Dans cette même catégorie, on peut inclure les jeux d'instructions vectorielles et les vectoriseurs qui étaient capables de reconnaître les boucles qui se prêtaient directement à une génération de code vectoriel. Au besoin le  
35 code source pouvait aussi être transformé pour faire apparaître des structures de code vectoriel.

- Une rupture dans les stratégies d'optimisation est venue avec l'introduction des pipelines, évolution architecturale qui découpe le traitement des instructions en plusieurs opérations exécutées séquentiellement, à l'image d'une chaîne d'assemblage dans une usine.
- 5 Cette technique qui permet de superposer l'exécution de plusieurs instructions à un instant donné améliore sensiblement les performances des systèmes, mais introduit des déviations importantes en cas de « rupture » de pipeline, suite par exemple à la présence d'instruction de branchement. Elle a ainsi mis fin à la prédictibilité du comportement d'un
- 10 code donné. Une autre rupture importante provient de l'utilisation des hiérarchies mémoires : l'optimisation devait alors s'appuyer sur l'évaluation fine d'une métrique très particulière : la localité (spatiale et temporelle). Cependant l'interaction entre le processeur et le système mémoire étant simple, le processus d'optimisation devait avoir pour principal objectif de
- 15 minimiser le nombre de "miss" (échecs) car chaque échec rajoutait une pénalité au temps d'exécution. Il faut noter cependant que la minimisation du nombre de "miss" était délicate et essentiellement réalisable pour des structures de code simple de type boucle. Cette difficulté d'estimation de manière statique des propriétés de localité conduisit à utiliser des
- 20 méthodes d'optimisation par gestion des profils : le code était exécuté une première fois pour déterminer précisément les propriétés de localité (construction d'un profil). Ce profil était alors utilisé dans une deuxième passe pour effectuer les optimisations liées à l'exploitation de la localité. A ce niveau de complexité architecturale, il était déjà très difficile de définir
- 25 de manière simple une stratégie efficace d'optimisation. Plus exactement, il était extrêmement délicat de savoir combiner différentes optimisations et ce même sur des cas très simples : il n'existait pas de mécanisme permettant de modéliser/prendre en compte de manière efficace les performances. C'est dans ce cadre qu'ont été développées les techniques
- 30 de compilation itérative qui combinent exécution et optimisation de manière à générer le meilleur code possible. Plus précisément, la compilation itérative consiste en la mise en œuvre d'une boucle de "transformation de code-mesure de performance (statique ou dynamique)". Il s'agit essentiellement de techniques permettant d'effectuer des transformations de code à partir de mesures de performance obtenues lors de l'exécution de programmes. Ces techniques sont
- 35



très élevé en temps de calcul et de mise au point de ces méthodes itératives a limité leur champ d'application à l'optimisation de bibliothèques.

5 Les architectures RISC ("Reduced Instruction Set Computer") qui  
utilisaient un jeu d'instructions simple et uniforme (à l'opposé du CISC)  
ont vu le jour vers le milieu des années 80. Les premières générations de  
processeurs RISC étaient très limitées en termes de fonctionnalité et la  
10 qualité du code généré (en particulier l'ordonnancement des instructions)  
devenait alors un facteur clé dans la course à la performance. De manière  
très similaire, les architectures VLIW ("Very Long Instruction Word")  
utilisaient des techniques de compilation tout à fait semblables pour  
exploiter au mieux le matériel. Ces deux architectures RISC et VLIW  
15 avaient toujours un modèle de performance simple déterministe dans la  
mesure où globalement les instructions s'exécutaient dans le même ordre  
que le code du programme généré, ce qui simplifiait considérablement le  
comportement de ces architectures et par là-même le processus  
d'optimisation. Toutefois, très rapidement ces architectures ont généralisé  
20 l'utilisation du pipeline et des caches mémoire pour obtenir de meilleures  
performances (plus précisément, ceci a amené une amélioration des  
performances en moyenne et au détriment de la prédictibilité)

L'apparition des architectures superscalaires (capables d'exécuter  
plusieurs instructions par cycle) et surtout les mécanismes de traitement  
25 dans le désordre ("Out of Order Processing") des instructions ont rendu  
encore plus difficiles les processus d'optimisation des performances. De  
plus, simultanément les hiérarchies mémoires ont évolué rapidement : le  
nombre de niveaux a augmenté et surtout divers mécanismes permettant  
de gérer de manière plus ou moins explicite les différents caches (pré-  
30 chargement, gestion de priorité,...) ont fait leur apparition. L'ensemble de  
ces mécanismes a rendu le comportement de fractions de code même  
très simples (boucle avec accès à 2 ou 3 tableaux) extrêmement difficile et  
par là-même impossible à optimiser en s'appuyant sur des modèles  
simples de performance. Cette situation n'a fait qu'empirer avec l'écart  
35 grandissant entre les performances des processeurs et celles des  
mémoires.

Globalement, au cours des deux dernières décennies, au niveau de l'architecture des processeurs, la richesse en termes de fonctionnalités s'est considérablement accrue. Ainsi le nombre de registres a considérablement augmenté : de 8 registres en standard sur les architectures CISC, on est passé à 32 registres sur les architectures RISC et même à plus de 80 registres sur les architectures superscalaires. Au premier abord on peut penser qu'augmenter le nombre de registres va simplifier l'optimisation. En pratique, l'évolution des mémoires a rendu l'utilisation des registres encore plus cruciale et sur ce problème d'allocation de registre, le coût des algorithmes efficaces d'allocation de registre a considérablement augmenté car leur complexité est exponentielle en fonction du nombre de registres disponibles.

En réponse à ces évolutions récentes, la technologie des compilateurs a peu évolué : le nombre d'optimisations mises en œuvre a bien sûr augmenté mais la capacité à définir une stratégie globale d'optimisation n'a pas progressé et a même diminué.

Enfin, une dernière tendance est apparue avec la compilation "dynamique". Le principe est simple et attrayant : la compilation dynamique (ou encore spécialisation à l'exécution) consiste à effectuer des optimisations de code au dernier moment, c'est-à-dire à l'exécution. Une séquence de code est adaptée ("spécialisée") en fonction des données d'entrée (contexte d'exécution) d'un programme. Un mécanisme à l'exécution "examine" le comportement des programmes suivant la fréquence d'exécution des séquences d'instructions et décide ou non de mettre en œuvre des versions optimisées pour un contexte d'exécution précis. Ce type de mécanisme intrinsèquement dynamique est limité à des techniques d'optimisation peu coûteuses en temps de calcul pour leur mise en œuvre car elles ne doivent pas pénaliser l'exécution des codes.

On rappelle par ailleurs qu'une bibliothèque est un ensemble de procédures, représentatif d'un domaine ou d'une portion de ce domaine : les composants d'une bibliothèque doivent en fait correspondre à des procédures standard fréquemment utilisées. Le concept de bibliothèque est très ancien, antérieur à la notion de compilateurs et c'est un des grands thèmes de la programmation. La bibliothèque est un ensemble de procédures qui sont utilisées par les programmes pour effectuer des opérations communes.

Les bibliothèques peuvent correspondre à différents niveaux d'abstraction du plus simple au plus compliqué : BLAS 1 ("Basic Linear Algebra Subroutines Level I") est un ensemble d'opérations très simples portant sur des vecteurs mais permettant d'exprimer un grand nombre d'algorithmes classiques d'algèbre linéaire. A l'autre extrême, LINPACK (resp. EISPACK) est un ensemble complet de procédures permettant la résolution de systèmes linéaires (resp. le calcul de vecteurs propres et de valeurs propres). Un grand nombre de bibliothèques ont été développées et largement utilisées dans les domaines particuliers suivants :

- Calcul scientifique : BLAS1, BLAS2, BLAS3, BLAST, SPARSE BLAS, LINPACK, LAPACK, BLACS, PVM, MPI, etc.
- Traitement de signal : FFTPACK, VSIPI, etc.
- Graphique : DirectX, OpenGL.

Le fait que, dans un domaine d'applications donné, un certain nombre de bibliothèques aient été définies et identifiées traduit le fait que ce domaine puisse être "synthétisé".

Le plus gros défaut des bibliothèques est cependant que leur fonctionnalité est très limitée et qu'elles induisent une utilisation manuelle (c.a.d. nécessitent l'insertion dans le code source d'un appel explicite de procédure).

Les premières générations de bibliothèques (correspondant à des procédures simples et de petite taille) ont été en général développées à la main en assembleur. Ceci est très souvent resté le cas dès que la performance est le critère majeur (sous réserve bien sûr que les procédures restent d'une taille raisonnable).

Par contre, à la différence de la compilation qui est globalement un processus "en ligne" (les temps de compilation doivent rester modérés), l'optimisation de bibliothèques est essentiellement un processus "hors connexion" et peut utiliser des méthodes beaucoup plus gourmandes en temps de calcul. Ainsi la compilation itérative est un excellent outil d'optimisation pour le développement de bibliothèques mais qui ne peut malheureusement être utilisé que pour des codes simples vu sa lourdeur d'utilisation.

Dans le même ordre d'idée, la technologie de type "Automatic Tuning Liner Algebra Software" permet de réaliser une partie de l'optimisation (choix des bons paramètres tels que tailles des blocs).

Malheureusement cette technique est d'utilisation très restreinte car elle est très dépendante du type d'application considéré (calcul sur des matrices denses, calculs caractérisés par une très forte localité temporelle).

- 5 Les outils d'analyse de performance existants sont très variés (en particulier en fonction de l'objectif recherché) :
- Tests de performances (« benchmarks ») : ce sont des codes plus ou moins représentatifs d'un domaine d'application et qui permettent de comparer les performances de diverses machines.
  - 10 • Simulateurs : ils permettent d'appréhender le comportement d'une architecture au niveau le plus fin. Malheureusement ils sont très coûteux, délicats à développer, très lents et pas nécessairement fidèles par rapport au processeur cible.
  - Modèles mathématiques : il s'agit de mettre en équation la performance de la machine. En général leur utilisation est extrêmement limitée et ils ne sont utiles que pour étudier différentes variantes simples autour d'un même code très simple.
  - 15 • Outils de contrôle/gestion de profils : ces outils permettent de récupérer (via des compteurs de matériel spécialisés) différentes informations relatives à l'exécution d'un programme : nombre de cycles, nombre de miss, etc.
  - 20

On peut par ailleurs formuler les remarques suivantes :

- Les tests de performance évoluent peu et surtout sont devenus l'objet d'enjeux commerciaux trop importants. Très souvent, leur optimisation acharnée par les constructeurs fausse la portée et la validité des résultats obtenus.
- 25 - Simulateurs : l'exploitation de leurs résultats (pour l'optimisation de code) devient de plus en plus difficile dans la mesure où l'architecture ciblée est devenue très complexe.
- 30 - Modèles mathématiques : ils évoluent peu et en dehors de l'usage local mentionné ci-dessus, ils sont inutilisables. Une des raisons en est que les bons outils mathématiques de modélisation présupposent un comportement "uniformément moyenné", ce qui est loin d'être le cas en pratique.

17 - "Outils de contrôle/gestion des profils" : ils fournissent essentiellement des informations relatives à l'exécution d'un programme et sont donc très utiles.

distribution de l'activité au cours du temps ; ils ne permettent pas de corréler code et comportement au niveau fin ; enfin, leur exploitation (comme dans le cas du simulateur) est très délicate, en raison notamment de la complexité de l'architecture ciblée.

5 La présente invention vise à remédier aux inconvénients précités et à permettre, pour une plate-forme matérielle prédéfinie comportant au moins un processeur, de générer de façon automatique des codes optimisés opérationnels sur cette plate-forme pour un domaine d'application prédéterminé à partir de codes sources soumis par des

10 utilisateurs.  
De façon plus particulière, l'invention vise à accroître les performances de systèmes informatiques de façon indépendante du choix du langage source et ceci pour des systèmes mettant en œuvre des processeurs dont l'architecture peut faire appel à des instructions simples  
15 ou complexes et qui peuvent comprendre un nombre plus ou moins important de registres, d'unités fonctionnelles et de niveaux de cache.

L'invention a également pour objet d'éviter les limitations de la couverture fonctionnelle des bibliothèques de programmes spécialisées et vise à créer un système de génération automatique de codes optimisés  
20 pour un grand nombre de structures de code voisines, qui peuvent présenter différents niveaux de complexité.

Ces buts sont atteints, conformément à l'invention, grâce à un système de génération automatique de codes optimisés opérationnels sur une plate-forme matérielle prédéfinie comportant au moins un processeur,  
25 pour un domaine d'application prédéterminé à partir de codes sources soumis par des utilisateurs, caractérisé en ce qu'il comprend des moyens de réception de séquences de codes symboliques dites séquences-étalons représentatives du comportement dudit processeur en termes de performances, pour le domaine d'application prédéterminé ; des moyens  
30 de réception de paramètres statiques définis à partir de la plate-forme matérielle prédéfinie, de son processeur et des séquences-étalons ; des moyens de réception de paramètres dynamiques également définis à partir de la plate-forme matérielle prédéfinie, de son processeur et des séquences-étalons ; un dispositif d'analyse pour établir des règles  
35 d'optimisation à partir de tests et de mesures de performances établis à partir des séquences-étalons, des paramètres statiques et des paramètres

dynamiques ; un dispositif d'optimisation et génération de code recevant d'une part les séquences-étalons et d'autre part les règles d'optimisation pour examiner les codes sources d'utilisateurs, détecter des boucles optimisables, décomposer en noyaux, assembler et injecter des codes pour délivrer des codes optimisés ; et des moyens pour réinjecter dans les séquences-étalons des informations issues du dispositif de génération et optimisation de codes et relatives à des noyaux.

De façon plus particulière, le dispositif d'analyse comprend un générateur de tests relié d'une part aux moyens de réception de séquences-étalons et d'autre part aux moyens de réception de paramètres statiques pour générer automatiquement un nombre élevé de variantes de tests qui sont transférées par des moyens de transfert pour être stockées dans une base des variantes ; un exerciceur relié d'une part à des moyens de transfert pour recevoir les variantes de tests stockées dans la base des variantes et d'autre part aux moyens de réception de paramètres dynamiques pour exécuter les variantes de tests dans une plage de variation des paramètres dynamiques et produire des résultats qui sont transférés par des moyens de transfert pour être stockés dans une base des résultats ; et un analyseur relié à des moyens de transfert pour recevoir les résultats stockés dans la base des résultats, analyser ceux-ci et en déduire des règles d'optimisation qui sont transférées par des moyens de transfert dans une base de règles d'optimisation.

Avantageusement, l'analyseur comprend des moyens de filtrage à un seuil arbitraire de la performance optimale, de manière à considérer une variante de la base des résultats comme optimale dans l'espace des paramètres dès lors qu'elle satisfait aux critères de filtrage.

Selon un mode de réalisation préférentiel, l'analyseur comprend en outre des moyens de modification des paramètres statiques et des moyens de modification des paramètres dynamiques.

Le dispositif d'optimisation et génération de code comprend un dispositif de génération de code optimisé et un optimiseur, ce dernier comprenant un module de choix de stratégie relié d'une part aux moyens de réception des noyaux identifiés dans les codes sources d'origine, d'autre part aux moyens de réception de séquences-étalons et d'autre part à des moyens de réception de règles d'optimisation pour générer des codes optimisés et des séquences-étalons optimisés.

5 pluralité de versions dont chacune est optimale pour une certaine combinaison de paramètres, et un module de combinaison et d'assemblage relié aux moyens de réception de règles d'optimisation, à des moyens de réception d'informations issues du module de choix de stratégie et à des moyens de réception de la pluralité des versions, pour délivrer à travers des moyens de transfert des informations comprenant les versions optimisées correspondantes, leur zone d'utilisation et le cas échéant le test à exécuter pour déterminer dynamiquement la version la plus adaptée.

10 Selon un mode de réalisation optionnel préférentiel, le système comprend une base des noyaux optimisés et le module de combinaison et d'assemblage est relié à la base des noyaux optimisés par des moyens de transfert pour stocker dans cette base des noyaux optimisés les informations comprenant les versions optimisées, leur zone d'utilisation et  
15 le cas échéant le test à exécuter pour déterminer dynamiquement la version la plus adaptée.

Le dispositif d'optimisation et génération de code comprend un optimiseur et un dispositif de génération de code optimisé, ce dernier comprenant un module de détection de boucles optimisables qui est relié à  
20 des moyens de réception des codes sources d'utilisateurs, un module de décomposition en noyaux, un module d'analyse de cas, d'assemblage et d'injection de code qui est relié à travers des moyens de transfert à l'optimiseur pour transmettre l'identification du noyau détecté et des moyens de transfert pour recevoir les informations décrivant le noyau  
25 optimisé correspondant, le module d'analyse de cas, d'assemblage et d'injection de code étant en outre relié à des moyens de fourniture des codes optimisés.

Le module d'analyse de cas, d'assemblage et d'injection de code peut en outre être relié à la base des noyaux optimisés pour recevoir  
30 les informations décrivant un noyau optimisé sans invoquer l'optimiseur si le noyau recherché y a été stocké.

Selon une caractéristique avantageuse, le module d'analyse de cas, d'assemblage et d'injection de code comprend en outre des moyens pour rajouter aux séquences-étalons des noyaux qui ont été mis en  
35 évidence dans ce module d'analyse de cas, d'assemblage et d'injection de

code sans avoir d'identification correspondante dans la base des noyaux optimisés, ni dans les séquences-étalons.

Selon un mode de réalisation particulier, le système comprend un compilateur et un éditeur de liens pour recevoir un code source remanié  
5 issu du dispositif d'optimisation et génération du code et produire un code binaire optimisé adapté à la plate-forme matérielle.

Le système peut comprendre des moyens pour transférer du code source des noyaux optimisés de la base des noyaux optimisés vers le compilateur.

10 Selon une autre variante de réalisation, le système peut comprendre un compilateur et un module d'installation pour installer sur la plate-forme matérielle une bibliothèque dynamique contenant l'ensemble des fonctionnalités des noyaux optimisés.

L'invention peut s'appliquer à différents domaines d'application  
15 et notamment au calcul scientifique, au traitement de signal et au traitement graphique.

Selon une caractéristique particulière, les séquences-étalons comprennent un ensemble de codes de type boucle simples et génériques spécifiés dans un langage de type source et organisés en niveaux de  
20 manière hiérarchique par ordre de complexité croissante du code du corps de boucle.

Dans ce cas, les séquences-étalons comprennent des séquences-étalons de niveau 0 dans lesquelles une seule opération élémentaire est testée et correspond à un corps de boucle constitué d'une  
25 seule expression arithmétique représentée par un arbre de hauteur 0.

En supplément, les séquences-étalons peuvent comprendre des séquences-étalons de niveau 1 pour lesquelles sont considérées et testées les combinaisons de deux opérations de niveau 0, les opérations des  
séquences-étalons de niveau 1 correspondant à des corps de boucles constitués soit d'une seule expression arithmétique représentée par un  
30 arbre de hauteur 1, soit de deux expressions arithmétiques, chacune étant représentée par un arbre de hauteur 0.

Selon un mode de réalisation possible, les séquences-étalons comprennent en outre des séquences-étalons de niveau 2 pour lesquelles  
35 sont considérées et testées les combinaisons de trois opérations de niveau 0, les opérations de niveau 2 correspondant à des corps de boucles constitués soit d'une seule expression arithmétique représentée par un arbre de hauteur 2, soit de deux expressions arithmétiques, chacune étant représentée par un arbre de hauteur 1.



Les paramètres statiques comprennent notamment le nombre d'itérations de la boucle pour chaque séquence-étalon, le pas d'accès aux tableaux et le type d'opérande, le type d'instructions utilisées, les stratégies de préchargement, les stratégies d'ordonnancement des instructions et des itérations

Les paramètres dynamiques comprennent notamment la localisation des opérandes tableaux dans les différents niveaux de la hiérarchie mémoire, la position relative des adresses de départ des tableaux et l'historique des branchements.

Avantageusement, la base des noyaux optimisés comprend des séquences de code source de type boucle correspondant à des fragments de code réel et utile et organisés en niveaux par ordre de complexité croissante.

La plate-forme matérielle prédéfinie peut comporter par exemple au moins un processeur du type dénommé Itanium™ de la société INTEL ou au moins un processeur du type Power ou Power PC™ de la société IBM.

Selon un mode de réalisation possible applicable plus particulièrement aux systèmes comportant un processeur de type ITANIUM™, les règles d'optimisation comprennent au moins certaines des règles suivantes :

- (a) minimiser le nombre d'Ecritures en cas de mauvaise performance des Ecritures par rapport aux Lectures,
- (b) importance de l'utilisation des paires de chargement en virgule flottante,
- (c) ajuster le degré de déroulage en fonction de la complexité du corps de boucle,
- (d) utiliser les latences opérationnelles des opérations arithmétiques,
- (e) appliquer des techniques de masquage pour les vecteurs courts,
- (f) utiliser des suffixes de localité pour les accès mémoire (Lecture, Ecriture, Préchargement),
- (g) définir les distances de Préchargement,

- (h) effectuer une vectorisation de degré 4 pour éviter une partie des conflits de bancs L2,
- (i) prendre en compte de multiples variantes pour éviter une autre partie des conflits de bancs L2 et les conflits dans la file d'attente des Lectures/Ecritures,
- (j) prendre en compte les gains de performances liés aux différentes optimisations,
- (k) utiliser des chaînes de branchement minimisant les mauvaises prédictions (vecteurs courts),
- (l) fusionner les accès mémoires (regroupement de pixels),
- (m) vectoriser les traitements sur pixels.

Selon un autre mode de réalisation possible applicable plus particulièrement aux systèmes comportant un processus de type Power ou Power PC™, les règles d'optimisation comprennent au moins certaines des règles suivantes :

- (a) réordonnancer les Lectures pour regrouper les défauts de caches,
- (b) utiliser le préchargement uniquement sur les Ecritures,
- (c) ajuster le degré de déroulage en fonction de la complexité du corps de boucle,
- (d) utiliser les latences opérationnelles des opérations arithmétiques,
- (e) utiliser des suffixes de localité pour les accès mémoire (Lecture, Ecriture, Préchargement),
- (f) définir les distances de Préchargement,
- (g) prendre en compte de multiples variantes pour éviter les conflits dans les files d'attente Lecture/Ecriture,
- (h) prendre en compte les gains de performances liés aux différentes optimisations.

On a vu que les techniques de programmation présentées ci-dessus permettent d'optimiser les performances des programmes exécutés sur les processeurs à architecture PowerPC.

- la Figure 1 est un schéma-bloc montrant l'ensemble des modules d'un système de génération automatique de codes optimisés selon l'invention,

5 - la Figure 2 est un schéma-bloc montrant de façon plus détaillée la constitution d'un module d'analyse de performances pouvant être mis en œuvre dans le système de la Figure 1,

- la Figure 3 est un schéma-bloc montrant de façon plus détaillée la constitution d'un module d'optimisation et de génération de code pouvant être mis en œuvre dans le système de la Figure 1,

10 - la Figure 4 est un schéma bloc montrant un premier exemple de module de génération de code source remanié, associé à des moyens d'obtention de code binaire optimisé pour la plate-forme cible concernée, et

15 - la Figure 5 est un schéma bloc montrant un deuxième exemple de module de génération de code source remanié, associé à des moyens d'obtention de code binaire optimisé pour la plate-forme cible concernée.

On se référera tout d'abord à la Figure 1 qui montre l'ensemble du système de génération automatique de code optimisé pour fournir, sur une sortie 73 d'un module 80 d'optimisation et de génération de code, des codes optimisés opérationnels sur une plate-forme matérielle 90 prédéfinie comportant au moins un processeur 91.

20 Le système de génération de code optimisé est adapté à un domaine d'application déterminé et reçoit, par une entrée 71 du module 80, des codes sources 17 soumis par des utilisateurs, le terme utilisateurs devant s'entendre au sens large et incluant notamment les utilisateurs finaux, les programmeurs d'applications ou les programmeurs systèmes.

25 Des séquences de code symboliques dites séquences-étalons 1 représentatives du comportement du processeur 91 en termes de performances pour le domaine d'application considéré, sont appliquées sur une entrée 52 du module 80 d'optimisation et de génération de code et sur une entrée 51 d'un module d'analyse 10.

30 L'analyse de l'effet des divers paramètres de l'environnement et de l'interaction entre séquences-étalons permet de situer les zones de bonnes et mauvaises performances et d'en appréhender les raisons. Les séquences-étalons ne représentent pas forcément des séquences de code réel généré par les langages de programmation classiques. Seul un sous-

ensemble des séquences-étalons testées correspond à des noyaux utilisés pour l'optimisation du code utilisateur.

Une boucle optimisable est une construction de programme codant la représentation algorithmique d'une opération plus ou moins complexe sur des variables-vecteurs.

Un noyau ou boucle élémentaire constitue une forme simple de boucle optimisable. Le module 80 du système selon l'invention permet de générer automatiquement un nombre de noyaux optimisés sensiblement supérieur au nombre de fonctions offertes par les bibliothèques spécialisées mathématiques. En général, plusieurs versions d'un même noyau peuvent être générées, chaque version étant optimisée pour une certaine combinaison de paramètres d'environnement.

La phase d'optimisation dans un optimiseur 12 (Figure 3) consiste ainsi en la génération automatique d'un ensemble ou bibliothèque de noyaux optimisés pour la plate-forme cible 90 représentant des fonctionnalités représentatives du domaine d'application.

La phase d'optimisation est associée à une phase de génération de code dans un générateur de code 18 (Figure 3) qui examine le code source du programme utilisateur pour y détecter les boucles optimisables afin de forcer l'utilisation de noyaux optimisés en lieu et place du code qu'aurait généré le compilateur standard.

Des moyens 74 sont prévus pour réinjecter dans les séquences-étalons 1 les informations issues du module 80.

Les phases d'optimisation et de génération de code sont précédées par une phase d'analyse dans un module d'analyse 10 qui sert à déterminer, pour la plate-forme matérielle cible 90 et le domaine d'application considéré, les règles d'optimisation à respecter pour obtenir des performances optimales. Une sortie 57 du module d'analyse 10 permet le transfert des règles d'optimisation vers une base 9 de règles d'optimisation elle-même reliée, par des moyens de transfert 18, à l'optimiseur 12 du module 80.

On décrira maintenant de façon plus particulière le module d'analyse 10 en référence à la Figure 2.

Des paramètres relatifs à la configuration matérielle et aux caractéristiques du compilateur sont fournis au module d'analyse 10 par le système d'exploitation du système d'ordinateur. Ces paramètres sont utilisés pour déterminer les règles d'optimisation à respecter pour obtenir des performances optimales.

l'optimisation, et également en fonction des séquences-étalons, sont appliqués par des moyens 53, 54 au module d'analyse 10.

5 Les paramètres statiques 2 peuvent comprendre notamment le nombre d'itérations de la boucle pour chaque séquence-étalon, le pas d'accès aux tableaux et le type d'opérande, le type d'instructions utilisées, les stratégies de préchargement, les stratégies d'ordonnancement des instructions et des itérations.

10 Les paramètres dynamiques 7 peuvent comprendre notamment la localisation des opérandes tableaux dans les différents niveaux de la hiérarchie mémoire, la position relative des adresses de départ des tableaux et l'historique des branchements.

Dans le module 10 d'analyse de performances, un générateur de test 3 exploite les données relatives aux paramètres statiques 2 et dynamiques 7, qui lui sont fournies par les entrées 51, 53 pour générer un nombre potentiellement très élevé de variantes qui sont transférées par des moyens de transfert 61 vers une base de données des variantes 4.

20 Un autre outil automatique 5 dénommé exerciceur reçoit, par des moyens de transfert 62, les données des variantes et la base des variantes 4, met en œuvre les tests ainsi fabriqués, les exécute en faisant varier dans leur plage de variation les paramètres dynamiques 7 fournis par les moyens de transfert 55 et transfère, par des moyens de transfert 63, les mesures pertinentes vers une autre base de données 6 dénommée base des résultats.

25 Les mesures stockées dans la base des résultats 6 sont elles-mêmes transférées, par des moyens de transfert 64, vers un analyseur 8 qui, à partir de l'identification des zones de bonne et mauvaise performance en fonction du paramétrage, permet la formulation des règles d'optimisation 9 qui sont transférées par les moyens de transfert 57 dans la base 9 de règles d'optimisation.

30 L'analyseur 8 comprend également des moyens 54 de modification des paramètres statiques 2 et des moyens 56 de modification des paramètres dynamiques, par exemple s'il a été constaté une faible sensibilité aux variations d'un paramètre donné.

35 L'analyseur 8 peut comprendre des moyens de filtrage à un seuil arbitraire de la performance optimale. Dans ce cas, une variante de la base des résultats qui ne correspond pas à des performances optimales

Selon une variante de réalisation avantageuse, le module 80 d'optimisation et de génération de code comprend une base des noyaux optimisés 16. Le module de combinaison et d'assemblage 14 est relié à la base des noyaux optimisés 16 et est configuré pour combiner les noyaux optimisés 16 avec les noyaux de base de données 12 pour générer des programmes optimisés.

les versions optimisées, leur zone d'utilisation et le cas échéant le test à exécuter pour déterminer dynamiquement la version la plus adaptée. Selon cette variante, le module 23 d'analyse de cas, d'assemblage et d'injection de code est en outre relié à la base 16 des noyaux optimisés par les moyens de transfert 72 pour recevoir les informations décrivant un  
5 noyau optimisé, sans invoquer l'optimiseur 12, si le noyau recherché a déjà été stocké dans cette base 16.

Comme on peut le voir sur la Figure 3, le module 23 d'analyse de cas, d'assemblage et d'injection de code comprend en outre des moyens  
10 74 pour rajouter aux séquences-étalons 1 des noyaux qui ont été mis en évidence dans ce module 23 sans avoir l'identification correspondante dans la base 16 des noyaux optimisés, ni dans les séquences-étalons.

La Figure 4 montre un exemple particulier de réalisation pour lequel on n'a pas représenté l'optimiseur 12 qui reste identique à la variante de  
15 la Figure 3 selon laquelle il existe une base 16 des noyaux optimisés.

Dans cet exemple de réalisation, le module 18 de génération de code produit en sortie 73 du module 23 d'analyse de cas, d'assemblage et d'injection de code, un code source remanié 19 qui est ensuite traité par des outils classiques de préparation de programme 81, 82 pour l'obtention  
20 de code binaire 83 optimisé pour la plate-forme cible 90.

La Figure 4 représente un exemple de réalisation qui peut être très facilement mis en œuvre. Le code source utilisateur d'origine 17 a été remanié au sein du module 80 d'optimisation et de génération de code déjà décrit plus haut, de telle manière que les boucles optimisables soient  
25 remplacées par des appels à des sous-programmes, le code correspondant à ces sous-programmes étant injecté dans le code source remanié 19 depuis la base des noyaux optimisés 16. Le code source 19 ainsi remanié contient alors tout le nécessaire pour générer un code binaire optimisé 83 adapté à la plate-forme matérielle 90, par un passage dans une chaîne  
30 classique comprenant un compilateur 81 et un éditeur de liens 82.

Selon une variante possible, du code source des noyaux optimisés de la base des noyaux optimisés 16 peut être utilisé directement dans l'étape de compilation en tant que bibliothèque source d'appoint. Ceci est symbolisé sur la Figure 4 par la flèche en pointillés 85 qui relie la base des  
35 noyaux optimisés 16 au compilateur 81. Cette variante permet ainsi

d'éviter l'injection directe de ce code source des noyaux optimisés dans le code source remanié et allège l'étape de génération au sein du module 18.

La Figure 5 représente une autre variante de réalisation de l'exemple de réalisation de la Figure 4.

- 5        La variante de la Figure 5 exploite la possibilité qu'offrent certains systèmes d'exploitation d'installer des bibliothèques sous forme de codes binaires exécutables et accessibles par des programmes par édition de liens dynamiques au moment de leur exécution.

- 10       Dans le cas de la variante de la Figure 5, il n'est pas non plus nécessaire d'injecter de code depuis la base des noyaux optimisés 16 vers le code source remanié 19. En revanche, une bibliothèque dynamique contenant l'ensemble des fonctionnalités des noyaux optimisés doit être installée sur la plate-forme cible 90 par l'intermédiaire d'un compilateur 181 et d'un module d'installation 182. On peut utiliser un seul compilateur 15 commun pour les compilateurs 81 et 181 de la Figure 5. Dans cette variante de la Figure 5, l'opération d'installation n'est nécessaire qu'une seule fois pour chaque plate-forme cible, de sorte que cette variante est la plus économe en termes de traitement global du processus d'optimisation.

- 20       La mise en œuvre d'un système de génération de code optimisé selon l'invention s'applique particulièrement bien aux trois domaines que sont le calcul scientifique, le traitement du signal et le traitement graphique.

- 25       En effet, les codes utilisés dans ces trois domaines présentent plusieurs caractéristiques CAR1 à CAR4 qui sont importantes pour cette mise en œuvre.

- CAR 1 : Les structures de type boucle (ou « nids de boucle ») constituent les portions de code les plus consommatrices en termes de temps d'exécution
- CAR 2 : Les structures de données utilisées sont en majeure partie 30 de type tableau multidimensionnel et sont accédées suivant des motifs très réguliers (lignes, colonnes, blocs, etc.)
- CAR 3 : Les boucles (ou nids de boucles) sont en général constituées d'itérations indépendantes et parallélisables



- CAR 4 : Le corps de la boucle est en général constitué d'une séquence d'expressions arithmétiques et correspond à un calcul uniforme (ou quasi-uniforme) sur un grand volume de données.

5 Naturellement, si ces trois domaines du calcul scientifique, du traitement de signal et du traitement graphique possèdent des points communs, ils présentent aussi certaines différences fortes. Ainsi dans le domaine du traitement de signal, les données de type complexe constituent un type de données extrêmement important qui requiert des optimisations spécifiques tandis que l'importance de ce type de données est marginale dans les deux autres domaines. Le traitement graphique quant à lui est fortement marqué par l'utilisation d'un type de données particulier, les pixels, et d'une arithmétique spéciale. De plus en graphique, la structuration des données et des algorithmes par rapport à un écran bidimensionnel est fondamentale.

10 Les quatre caractéristiques ci-dessus (CAR1 à CAR4) ont des conséquences très fortes pour l'optimisation de code et permettent le développement de techniques complètement spécifiques:

- CAR 1  $\Rightarrow$  les optimisations vont en fait se concentrer sur les structures de type « boucle » qui présentent deux avantages majeurs : répétitivité (et prédictibilité) et compacité de représentation.
- CAR 2  $\Rightarrow$  les accès aux tableaux qui représentent une part importante (voire majeure en raison de l'utilisation croissante des caches) des temps d'exécution vont pouvoir facilement être analysés et optimisés en raison de leur régularité.
- CAR 3  $\Rightarrow$  l'indépendance des itérations au sein d'une boucle et de nids de boucles, va permettre l'utilisation (l'optimisation) de parcours de l'espace d'itération en fonction des accès aux tableaux et ce suivant les caractéristiques propres de l'architecture cible. On peut remarquer qu'accéder à N éléments donnés d'un tableau peut être réalisé de N ! manières (ordres) différentes.

- CAR 4  $\Rightarrow$  la structure simple du corps de boucles en termes d'expressions arithmétiques va nous permettre d'utiliser une approche systématique et hiérarchique s'appuyant sur la représentation sous forme d'arbres d'une expression arithmétique.

5

La phase d'analyse reste une phase essentiellement expérimentale à l'issue de laquelle il faut :

- avoir déterminé les points forts et les points faibles de l'architecture,
- 10 • savoir corréler performance et structure de code,
- avoir identifié les bonnes stratégies d'optimisation, celles ci pouvant être fonction de différents paramètres liés au code.

Comme on l'a déjà indiqué, le point de départ est un ensemble de  
15 codes « de type source » simples mais génériques appelés « séquences-étalons ». Ces codes ont une structure de type boucle, le terme « type source » indiquant que les opérations sont spécifiées à haut niveau et non au niveau assembleur.

Ces codes sont organisés en niveaux de manière hiérarchique par  
20 ordre de complexité croissante du code du corps de boucle de la manière suivante :

- SEQUENCE-ETALON NIVEAU 0 : à ce niveau, une seule opération élémentaire est testée, c'est-à-dire que le corps de boucle comporte une seule opération : lecture d'un élément de tableau, écriture d'un  
25 élément de tableau, addition flottante, etc. Ces opérations correspondent à des corps de boucle constitué d'une seule expression arithmétique représentée par un arbre de hauteur 0.
- SEQUENCE-ETALON NIVEAU 1 : à ce niveau, sont considérées et testées les combinaisons de deux opérations de niveau 0 : lecture  
30 et écriture d'un tableau, lecture de deux tableaux différents, lecture et écriture sur un tableau etc. Ces opérations correspondent à des corps de boucle constitués d'une seule expression

arithmétique représentée par un arbre de hauteur 1, soit deux expressions arithmétiques, chacune étant représentée par un arbre de hauteur 0.

- 5     • SEQUENCE-ETALON NIVEAU 2 : à ce niveau les combinaisons de deux opérations de niveau 1 ou de trois opérations de niveau 0 sont considérées et testées : lecture de trois tableaux différents, lecture et addition de deux tableaux composante à composante et écriture du résultat dans un troisième tableau etc.
- 10    • SEQUENCE-ETALON NIVEAU K : le niveau K peut être aisément défini par récurrence à partir des niveaux précédents.

Toutes les séquences-étalons de niveau 0 correspondent à des codes qui sont « artificiels », i.e. ne traduisent pas des boucles « réelles ».

Cette organisation en niveaux de complexité croissante est aussi utilisée dans la phase d'optimisation.

- 15    L'ensemble des séquences-étalons ainsi défini est infini.

Ces séquences-étalons utilisent deux classes de paramètres différents :

- 20    • Paramètres statiques : ces paramètres sont définis de manière statique (c'est-à-dire avant exécution et indépendamment de l'exécution). Ces paramètres statiques sont eux mêmes subdivisés en deux grandes sous classes : paramètres statiques de haut niveau (nombre d'itérations de la boucle, pas d'accès aux tableaux, type d'opérande,...), paramètres statiques de bas niveau (utilisation d'instructions spécifiques, ordonnancement des instructions etc.)
- 25    • Paramètres dynamiques : ces paramètres sont définis lors de l'exécution de la boucle. Ils comprennent par exemple : localisation des opérandes tableaux dans la hiérarchie mémoire, position relative des adresses de départ des tableaux,...

30    Ces deux classes de paramètres sont exploitées de manière très différente : les paramètres statiques seront utilisés pour générer les

différents codes tests en combinaison avec les variantes/optimisations décrites ci-dessous, tandis que les paramètres dynamiques sont exclusivement utilisés lors de l'exécution sur le banc de test.

- 5 Les paramètres statiques de haut niveau sont relativement limités et correspondent essentiellement aux paramètres classiques d'une boucle et de tableaux exprimés dans un langage haut niveau (Fortran ou C par exemple) sans aucune spécificité provenant du processeur cible.

- 10 Les paramètres statiques de bas niveau vont permettre de prendre en compte toutes les spécificités liées au processeur (architecture) et à l'ordonnancement des instructions (générateur de code objet). En effet, les séquences-étalons sont de fait des abstractions à haut niveau (définies dans un langage source, indépendantes de l'architecture du processeur visé) et n'intègrent en particulier aucune optimisation. Pour les tester sur un processeur donné, il faut générer et optimiser les codes assembleurs  
15 correspondants. Lors de cette génération, plusieurs variantes (séquence d'instructions assembleurs) vont être générées automatiquement. Toutes les variantes associées à une même séquence-étalon sont des codes sémantiquement équivalents à la séquence-étalon de départ. Ce sont ces variantes qui vont être exécutées et mesurées. Ces variantes  
20 correspondent en fait à différentes techniques d'optimisation de code (c'est-à-dire à des paramètres statiques de bas niveau). Ces optimisations peuvent être définies de manière abstraite sans référence à la structure particulière d'une séquence-étalon et constituent l'essentiel des paramètres statiques de bas niveau.

25

Les paramètres statiques de bas niveau comportent :

- l'utilisation des instructions assembleurs: une même opération au niveau source peut être mise en oeuvre par différentes séquences d'instructions. En particulier, il faut traiter ici les différentes  
30 stratégies possibles pour utiliser le pré-chargement des données et les instructions.

- la structure du corps de boucle : déroulage (degré variable) du corps de boucle,
- l'ordonnancement du corps de boucle : ordonnancement des instructions du corps de boucle (distances de préchargement, vectorisation), regroupement des défaut de cache, traitement des conflits dans le files d'attente),
- l'ordonnancement des itérations : pipeline logiciel (profondeur variable)

10 Dans nombre de compilateurs, les paramètres statiques de bas niveau décrits ci-dessus correspondent à des options de compilation permettant de mettre en œuvre de manière explicite l'optimisation visée.

Le rôle du générateur de test 3 est de générer les différentes variantes décrites ci-dessus correspondant d'une part aux paramètres statiques de haut niveau (par exemple, pas d'accès aux tableaux) et d'autre part aux paramètres statiques de bas niveau.

Il faut noter que pour des séquences-étalons de niveau 1, le nombre total de variantes à générer/analyser est extrêmement élevé et se chiffre à plusieurs millions. Malgré cela, le processus de génération et d'analyse peut être très simplement automatisé.

Au niveau de l'exerciceur 5 et de l'analyseur 8, il s'agit de tester les performances des différentes variantes et de sélectionner les meilleures variantes/optimisations possibles.

Cette phase implique la génération d'un grand nombre de résultats stockés dans la base de résultats 6. Les expérimentations sont effectuées de manière hiérarchique et entrelacée avec les phases d'analyse : ainsi les premières expérimentations sont effectuées sur les variantes des séquences-étalons de niveau 0. A l'issue de cette première campagne d'expérimentation, un premier tri sur les différentes variantes peut être effectué en fonction des résultats obtenus. Certaines variantes peuvent ainsi être tout de suite éliminées et ne seront plus considérées pour les

niveaux suivants. Ceci permet de limiter l'explosion combinatoire du nombre d'expériences à réaliser.

La phase d'analyse des résultats est à première vue assez simple à effectuer car une seule métrique (performance) est utilisée. En fait une grande partie de la complexité du processus provient du fait qu'en général, le choix des meilleures variantes va très fortement dépendre des paramètres.

Un premier tri peut être réalisé de manière très simple en calculant d'après les spécifications de l'architecture, les performances optimales pour chacune des séquences-étalons. Par contre, des difficultés peuvent apparaître rapidement, liées aux interactions complexes entre architecture et codes (y compris pour des codes aussi simples que les séquences-étalons de niveau 0 et 1): ceci se traduit par des figures compliquées décrivant les variations de la performance en fonction des paramètres. Ces comportements complexes peuvent être d'abord analysés en utilisant des algorithmes de traitement d'image puis synthétisés en qualifiant une variante donnée pour une certaine plage de paramètre. Ainsi la phase d'analyse ne génère pas simplement une liste indiquant pour chaque séquence-étalon la meilleure (et unique) variante/technique d'optimisation : de fait pour chaque séquence-étalon, une liste de plages de paramètres est déterminée et, pour chacune de ces plages, la meilleure variante/technique d'optimisation est indiquée : c'est une telle information qui sera appelée « règle d'optimisation ».

L'ensemble des séquences-étalons testées est un sous-ensemble très restreint de l'ensemble des séquences-étalons. Cet ensemble qui est utilisé par la suite pour les optimisations est appelé « ensemble des séquences-étalons de référence ».

En pratique, il est très important de fixer un objectif « raisonnable » d'optimisation : ainsi chercher à tout prix l'optimum peut générer un nombre très élevé de variantes alors qu'en relâchant la contrainte d'optimum et en se contentant de performances dans la

pour une grande plage de paramètres. On procède pour cela à un filtrage par exemple à un seuil de 90% de la performance optimale.

En pratique, il suffit de tester et d'analyser seulement les niveaux 0, 1 et 2 des séquences-étalons pour dégager et valider les principales techniques d'optimisation. L'ensemble de référence des séquences-étalons ne contiendra pas en général de séquence de niveau supérieur à 3.

En effet, le volume d'expérimentations devient rapidement très élevé en particulier au-delà du niveau 2.

L'ensemble des expérimentations se prête de manière idéale à la parallélisation : les tests peuvent être exécutés en parallèle sur 100 ou 1000 machines. Cette propriété de parallélisation est extrêmement utile et permet d'effectuer des explorations systématiques et ceci dans des temps acceptables.

Cette phase est entièrement automatisable et les procédures de vérification de qualité et de cohérence des résultats peuvent être aussi automatisées. L'intervention humaine n'est requise que pour le relevé d'erreurs/anomalies découlant de l'analyse des résultats produits automatiquement par les procédures de vérification de qualité et de cohérence.

A l'issue de la phase d'analyse, l'objectif est de disposer d'un très grand nombre de codes simples dits « noyaux » spécifiquement optimisés pour l'architecture cible, ce processus d'optimisation s'appuyant de manière essentielle sur les techniques d'optimisation dégagées à l'issue de la phase d'analyse.

De manière stricte, les « noyaux » sont des séquences de code source de type boucle et constituent un sous-ensemble du cas général des séquences-étalons. A la différence de ces dernières, ils correspondent à des fragments de code réel et utile. Comme les séquences-étalons, ils sont organisés en niveaux par ordre de complexité croissante.

La génération/optimisation de ces noyaux se déroule suivant les quatre phases ci dessous :

- 5           ○ Corrélation avec une ou plusieurs séquences-étalons de référence : pour les noyaux les plus simples, il y a une correspondance directe entre noyaux et séquences-étalons, pour les noyaux plus complexes, le noyau sera décomposé en plusieurs séquences-étalons de référence. Cette corrélation/décomposition est effectuée au niveau source en fonction des caractéristiques du corps de boucle du noyau : nombre de tableaux, pas d'accès aux tableaux, etc...
- 10          ○ Génération de code/Ordonnancement d'instructions/optimisation d'instructions : il s'agit d'utiliser les techniques d'optimisation détectées lors de la phase d'analyse (en fonction des séquences-étalons correspondantes) et de les appliquer directement à la génération / optimisation de code pour les noyaux. Pour un même noyau, plusieurs versions possibles pourront être générées et ce en
- 15           fonction des paramètres.
- Allocations de registre : bon nombre des techniques d'optimisation utilisées accroissent de manière sensible la pression sur les registres disponibles. Dans ce cas, il convient d'aménager l'allocation de l'ensemble des registres disponibles.
- 20          ○ Expérimentation/Validation : le noyau généré et optimisé est testé en utilisant le banc de test de la phase d'analyse. A l'issue de cette phase, un modèle simple des performances du noyau est construit.

Par rapport aux optimisations classiques utilisées dans un compilateur, les optimisations utilisées ici sont très différentes : tout

25 d'abord elles sont dérivées directement d'un processus détaillé d'évaluation de performances (effectué lors de la phase d'analyse) ensuite elles sont beaucoup plus complexes et performantes (en particulier pour l'allocation de registres) car elles sont exécutées hors connexion sans « contraintes » de temps.

30 L'utilisation des séquences-étalons de référence et des règles de génération permettent de prendre en compte d'une part toutes les caractéristiques des noyaux et d'autre part les contraintes matérielles



mesurées et non théoriques) et d'autre part les différentes versions devant être sélectionnées en fonction des paramètres.

5 A l'issue de cette phase, il a été construit une base de données 16 des noyaux optimisés, contenant non seulement les codes générés mais aussi les informations relatives aux performances et ceci en fonction des différents paramètres. Chacun de ces noyaux est d'ailleurs testé suivant la procédure utilisée pour les séquences-étalons.

10 En pratique, la base 16 des noyaux optimisés comprend de manière systématique et exhaustive tous les noyaux de niveaux 1, 2, 3, 4 et 5. Le coût en termes de volume de calcul de la construction de cette base de données est important mais comme pour la phase analyse de performances, elle se parallélise très efficacement.

L'optimisation du code utilisateur se déroule en trois phases :

- 15     ○ Détection des boucles optimisables (module 20) : il s'agit de reconnaître au sein du code source les boucles pouvant être décomposées en noyaux. Cette phase utilise des techniques très proches de celle des paralléliseurs/vectoriseurs automatiques. Au besoin, le code source est restructuré pour faire apparaître la boucle sous la forme la plus favorable à l'optimisation
- 20     ○ Analyse de la boucle optimisable, décomposition en noyaux (module 22) : en s'appuyant sur des techniques d'appariement de configuration et de décomposition proches de celles utilisées pour l'optimisation des noyaux, la boucle est décomposée en une séquence de noyaux
- 25     ○ Assemblage et injection de code (module 23) : les différents noyaux utilisés pour la décomposition sont assemblés et réinjectés dans le code source.

La procédure de décomposition est en général paramétrée en fonction de caractéristiques de la boucle source d'origine.

30 Les optimisations proposées peuvent être intégrées

- soit par un pré-processeur dans un chaîne de compilation existante et ceci de manière transparente c'est-à-dire sans qu'il soit nécessaire d'avoir accès au code du compilateur,
- soit directement dans un compilateur, ceci nécessitant bien sûr des modifications dans le code du compilateur.

Comme on l'a déjà indiqué plus haut en référence à la Figure 3, à l'issue de la phase d'analyse un certain nombre de règles d'optimisation sont disponibles : ces règles sont fonction de la séquence-étalon et de la  
10 plage de paramètre. Au lieu de passer par l'étape intermédiaire des noyaux, une variante possible est de corrélér directement la boucle optimisable avec les séquences-étalons et d'appliquer directement les règles d'optimisation sur la boucle optimisable sans passer par le stockage de noyaux dans une base 16 des noyaux optimisés.

15 Cette variante est plus simple que le passage par les noyaux et elle permet une utilisation plus souple des règles d'optimisation. En revanche, du fait qu'elle est réalisée essentiellement en ligne, le nombre de variantes explorées sera nécessairement plus restreint et donc les performances obtenues seront a priori un peu moins bonnes.

20 A l'issue de la phase d'optimisation, le système a généré des formes optimisées pour un certain nombre de boucles « optimisables » qui a priori n'étaient pas directement disponibles dans la base des noyaux car elles avaient nécessité une décomposition. Ces formes optimisées peuvent être stockées dans la base 16 des noyaux optimisés et réutilisées par la  
25 suite. Ainsi, la base de données 16 des noyaux est enrichie automatiquement par cette forme d'apprentissage.

REVENDEICATIONS

1. Système de génération automatique de codes optimisés (19)  
opérationnels sur une plate-forme matérielle prédéfinie (90) comportant  
5 au moins un processeur (91), pour un domaine d'application prédéterminé  
à partir de codes sources (17) soumis par des utilisateurs, caractérisé en  
ce qu'il comprend des moyens (51, 52) de réception de séquences de  
codes symboliques dites séquences-étalons (1) représentatives du  
comportement dudit processeur (91) en termes de performances, pour le  
10 domaine d'application prédéterminé ; des moyens (53) de réception de  
paramètres statiques (2) définis à partir de la plate-forme matérielle  
prédéfinie (90), de son processeur (91) et des séquences-étalons (1) ; des  
moyens (55) de réception de paramètres dynamiques (7) également  
définis à partir de la plate-forme matérielle prédéfinie (90), de son  
15 processeur (91) et des séquences-étalons (1) ; un dispositif d'analyse (10)  
pour établir des règles d'optimisation (9) à partir de tests et de mesures  
de performances établis à partir des séquences-étalons (1), des  
paramètres statiques (2) et des paramètres dynamiques (7) ; un dispositif  
(80) d'optimisation et génération de code recevant d'une part les  
20 séquences-étalons (1) et d'autre part les règles d'optimisation (9) pour  
examiner les codes sources (17) d'utilisateurs, détecter des boucles  
optimisables, décomposer en noyaux, assembler et injecter des codes  
pour délivrer des codes optimisés (19) ; et des moyens (74) pour  
réinjecter dans les séquences-étalons (1) des informations issues du  
25 dispositif (80) de génération et optimisation de codes et relatives à des  
noyaux .

2. Système selon la revendication 1, caractérisé en ce que le  
dispositif d'analyse (10) comprend un générateur de tests (3) relié d'une  
part aux moyens (51) de réception de séquences-étalons et d'autre part  
30 aux moyens (53) de réception de paramètres statiques pour générer  
automatiquement un nombre élevé de variantes de tests qui sont  
transférées par des moyens de transfert (61) pour être stockées dans une  
base des variantes (4) ; un exerciceur (5) relié d'une part à des moyens  
de transfert (62) pour recevoir les variantes de tests stockées dans la base  
35 des variantes (4) et d'autre part aux moyens (55) de réception de  
paramètres dynamiques pour exécuter les variantes de tests dans une

plage de variation des paramètres dynamiques (7) et produire des résultats qui sont transférés par des moyens de transfert (63) pour être stockés dans une base des résultats (6) ; et un analyseur (8) relié à des moyens de transfert (64) pour recevoir les résultats stockés dans la base des résultats (6), analyser ceux-ci et en déduire des règles d'optimisation qui sont transférées par des moyens de transfert (57) dans une base de règles d'optimisation (9).

3. Système selon la revendication 2, caractérisé en ce que l'analyseur (8) comprend des moyens de filtrage à un seuil arbitraire de la performance optimale, de manière à considérer une variante de la base des résultats comme optimale dans l'espace des paramètres dès lors qu'elle satisfait aux critères de filtrage.

4. Système selon la revendication 2 ou la revendication 3, caractérisé en ce que l'analyseur (8) comprend en outre des moyens (54) de modification des paramètres statiques (2) et des moyens (56) de modification des paramètres dynamiques (7).

5. Système selon l'une quelconque des revendications 1 à 4, caractérisé en ce que le dispositif (80) d'optimisation et génération de code comprend un dispositif (18) de génération de code optimisé et un optimiseur (12), ce dernier comprenant un module de choix de stratégie (13) relié d'une part aux moyens (92) de réception des noyaux identifiés dans les codes sources d'origine, d'autre part aux moyens (52) de réception de séquences-étalons (1) et d'autre part à des moyens (58) de réception de règles d'optimisation (9) pour générer, pour chaque noyau correspondant à une séquence-étalon testée une pluralité de versions (15) dont chacune est optimale pour une certaine combinaison de paramètres, et un module de combinaison et d'assemblage (14) relié aux moyens (59) de réception de règles d'optimisation (9), à des moyens (66) de réception d'informations issues du module de choix de stratégie (13) et à des moyens (68) de réception de la pluralité des versions (15), pour délivrer à travers des moyens de transfert (93) des informations comprenant les versions optimisées correspondantes, leur zone d'utilisation et le cas échéant le test à exécuter pour déterminer dynamiquement la version la plus adaptée.

6. Système selon la revendication 5, caractérisé en ce qu'il comprend une base de données pour stocker les informations relatives aux versions optimisées et leur zone d'utilisation.

combinaison et d'assemblage (14) est relié à la base des noyaux optimisés (16) par des moyens de transfert (79) pour stocker dans cette base des noyaux optimisés les informations comprenant les versions optimisées, leur zone d'utilisation et le cas échéant le test à exécuter pour déterminer dynamiquement la version la plus adaptée.

7. Système selon l'une quelconque des revendications 1 à 6, caractérisé en ce que le dispositif (80) d'optimisation et génération de code comprend un optimiseur (12) et un dispositif (18) de génération de code optimisé, ce dernier comprenant un module (20) de détection de boucles optimisables qui est relié à des moyens (71) de réception des codes sources (17) d'utilisateurs, un module (22) de décomposition en noyaux, un module (23) d'analyse de cas, d'assemblage et d'injection de code qui est relié à travers des moyens de transfert (92) à l'optimiseur (12) pour transmettre l'identification du noyau détecté et des moyens de transfert (93) pour recevoir les informations décrivant le noyau optimisé correspondant, le module (23) d'analyse de cas, d'assemblage et d'injection de code étant en outre relié à des moyens (73) de fourniture des codes optimisés.

8. Système selon les revendications 6 et 7, caractérisé en ce que le module (23) d'analyse de cas, d'assemblage et d'injection de code est en outre relié à la base (16) des noyaux optimisés pour recevoir les informations décrivant un noyau optimisé sans invoquer l'optimiseur (12) si le noyau recherché y a été stocké.

9. Système selon la revendication 8, caractérisé en ce que le module (23) d'analyse de cas, d'assemblage et d'injection de code comprend en outre des moyens (74) pour rajouter aux séquences-étalons (1) des noyaux qui ont été mis en évidence dans ce module (23) d'analyse de cas, d'assemblage et d'injection de code sans avoir d'identification correspondante dans la base (16) des noyaux optimisés, ni dans les séquences-étalons.

10. Système selon l'une quelconque des revendications 6, 8 et 9, caractérisé en ce qu'il comprend un compilateur (81) et un éditeur de liens (82) pour recevoir un code source remanié (19) issu du dispositif (80) d'optimisation et génération du code et produire un code binaire optimisé (83) adapté à la plate-forme matérielle (90).

11. Système selon la revendication 10, caractérisé en ce qu'il comprend des moyens (85) pour transférer du code source des noyaux optimisés de la base des noyaux optimisés (16) vers le compilateur (81).

12. Système selon la revendication 10, caractérisé en ce qu'il comprend un compilateur (181) et un module d'installation (182) pour installer sur la plate-forme matérielle (90) une bibliothèque dynamique contenant l'ensemble des fonctionnalités des noyaux optimisés.

13. Système selon l'une quelconque des revendications 1 à 12, caractérisé en ce que le domaine d'application prédéterminé est le calcul scientifique.

14. Système selon l'une quelconque des revendications 1 à 12, caractérisé en ce que le domaine d'application prédéterminé est le traitement du signal.

15. Système selon l'une quelconque des revendications 1 à 12, caractérisé en ce que le domaine d'application prédéterminé est le traitement graphique

16. Système selon l'une quelconque des revendications 1 à 15, caractérisé en ce que les séquences-étalons (1) comprennent un ensemble de codes de type boucle simples et génériques spécifiés dans un langage de type source et organisés en niveaux de manière hiérarchique par ordre de complexité croissante du code du corps de boucle.

17. Système selon la revendication 16, caractérisé en ce que les séquences-étalons (1) comprennent des séquences-étalons de niveau 0 dans lesquelles une seule opération élémentaire est testée et correspond à un corps de boucle constitué d'une seule expression arithmétique représentée par un arbre de hauteur 0.

18. Système selon la revendication 17, caractérisé en ce que les séquences-étalons comprennent des séquences-étalons de niveau 1 pour lesquelles sont considérées et testées les combinaisons de deux opérations de niveau 0, les opérations des séquences-étalons de niveau 1 correspondant à des corps de boucles constitués soit d'une seule expression arithmétique représentée par un arbre de hauteur 1, soit de deux expressions arithmétiques, chacune étant représentée par un arbre de hauteur 1.

19. Système selon la revendication 18, caractérisé en ce que les séquences-étalons de niveau 1 comprennent des séquences-étalons de niveau 1.1 pour lesquelles sont considérées et testées les combinaisons de deux opérations de niveau 0, les opérations des séquences-étalons de niveau 1.1 correspondant à des corps de boucles constitués soit d'une seule expression arithmétique représentée par un arbre de hauteur 1, soit de deux expressions arithmétiques, chacune étant représentée par un arbre de hauteur 1.

2 pour lesquelles sont considérées et testées les combinaisons de deux opérations de niveau 1 ou de trois opérations de niveau 0.

5 20. Système selon l'une quelconque des revendications 16 à 19, caractérisé en ce que les paramètres statiques (2) comprennent notamment le nombre d'itérations de la boucle pour chaque séquence-étalon, le pas d'accès aux tableaux et le type d'opérande, le type d'instructions utilisées, les stratégies de préchargement, les stratégies d'ordonnancement des instructions et des itérations.

10 21. Système selon l'une quelconque des revendications 16 à 20, caractérisé en ce que les paramètres dynamiques (7) comprennent notamment la localisation des opérandes tableaux dans les différents niveaux de la hiérarchie mémoire, la position relative des adresses de départ des tableaux et l'historique des branchements.

15 22. Système selon l'une quelconque des revendications 6, 8 et 9, caractérisé en ce que la base des noyaux optimisés (16) comprend des séquences de code source de type boucle correspondant à des fragments de code réel et utile et organisés en niveaux par ordre de complexité croissante.

20 23. Système selon l'une quelconque des revendications 1 à 12, caractérisé en ce que la plate-forme matérielle prédéfinie (90) comporte au moins un processeur du type Itanium.

24. Système selon l'une quelconque des revendications 1 à 12, caractérisé en ce que la plate-forme matérielle prédéfinie (90) comporte au moins un processeur du type Power ou PowerPC.

25 25. Système selon l'une quelconque des revendications 13 à 15 et selon la revendication 23, caractérisé en ce que les règles d'optimisation (9) comprennent au moins certaines des règles suivantes :

- (a) minimiser le nombre d'Ecritures en cas de mauvaise performance des Ecritures par rapport aux Lectures,
- 30 (b) importance de l'utilisation des paires de chargement en virgule flottante,
- (c) ajuster le degré de déroulage en fonction de la complexité du corps de boucle,
- (d) utiliser les latences opérationnelles des opérations arithmétiques,

- (e) appliquer des techniques de masquage pour les vecteurs courts,
- (f) utiliser des suffixes de localité pour les accès mémoire (Lecture, Ecriture, Préchargement),
- (g) définir les distances de Préchargement,
- 5 (h) effectuer une vectorisation de degré 4 pour éviter une partie des conflits de bancs L2
- (i) prendre en compte de multiples variantes pour éviter une autre partie des conflits de bancs L2 et les conflits dans la file d'attente des Lectures/Ecritures
- 10 (j) prendre en compte les gains de performances liés aux différentes optimisations.
- (k) utiliser des chaînes de branchement minimisant les mauvaises prédictions (vecteurs courts)
- (l) fusionner les accès mémoires (regroupement de pixels)
- 15 (m) vectoriser les traitements sur pixels.

26. Système selon l'une quelconque des revendications 13 à 15 et selon la revendication 24, caractérisé en ce que les règles d'optimisation (9) comprennent au moins certaines des règles suivantes :

- (a) réordonnancer les Lectures pour regrouper les défauts de caches
- 20 (b) utiliser le préchargement uniquement sur les Ecritures
- (c) ajuster le degré de déroulage en fonction de la complexité du corps de boucle,
- (d) utiliser les latences opérationnelles des opérations arithmétiques,
- (e) utiliser des suffixes de localité pour les accès mémoire (Lecture, Ecriture, Préchargement),
- 25 (f) définir les distances de Préchargement,
- (g) prendre en compte de multiples variantes pour éviter les conflits dans les files d'attente Lecture/Ecriture

27. Système selon l'une quelconque des revendications 13 à 15 et selon la revendication 24, caractérisé en ce que les règles d'optimisation (9) comprennent au moins certaines des règles suivantes :



1/5

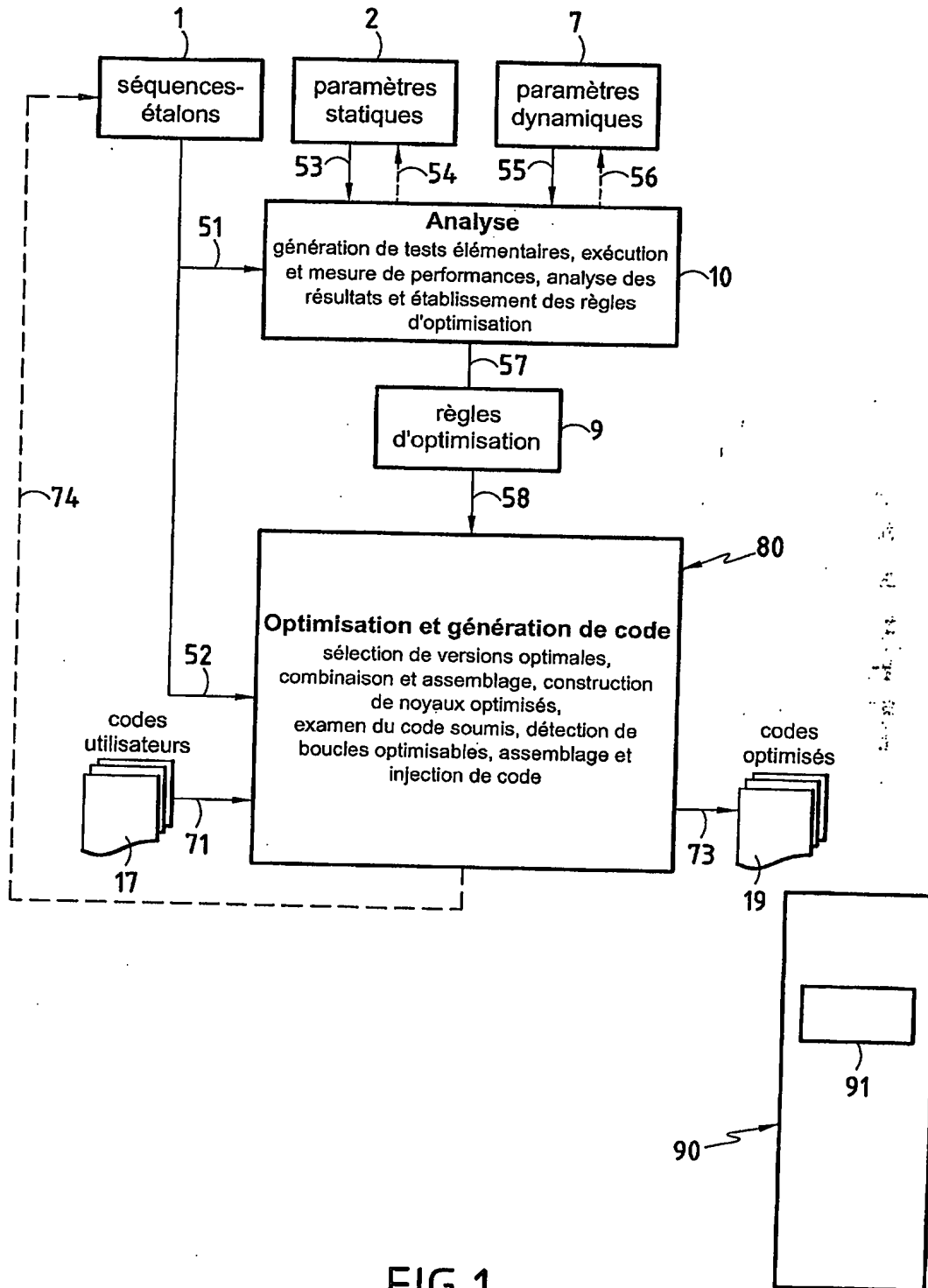


FIG.1

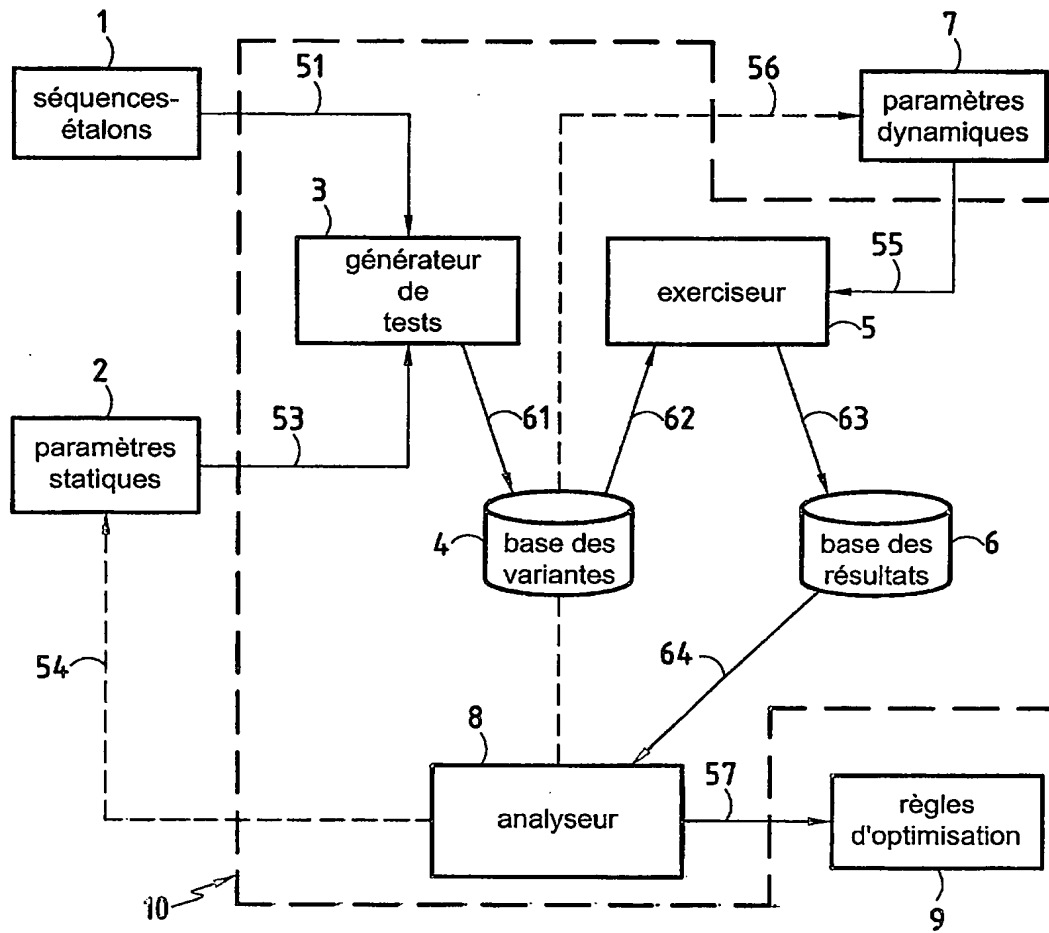
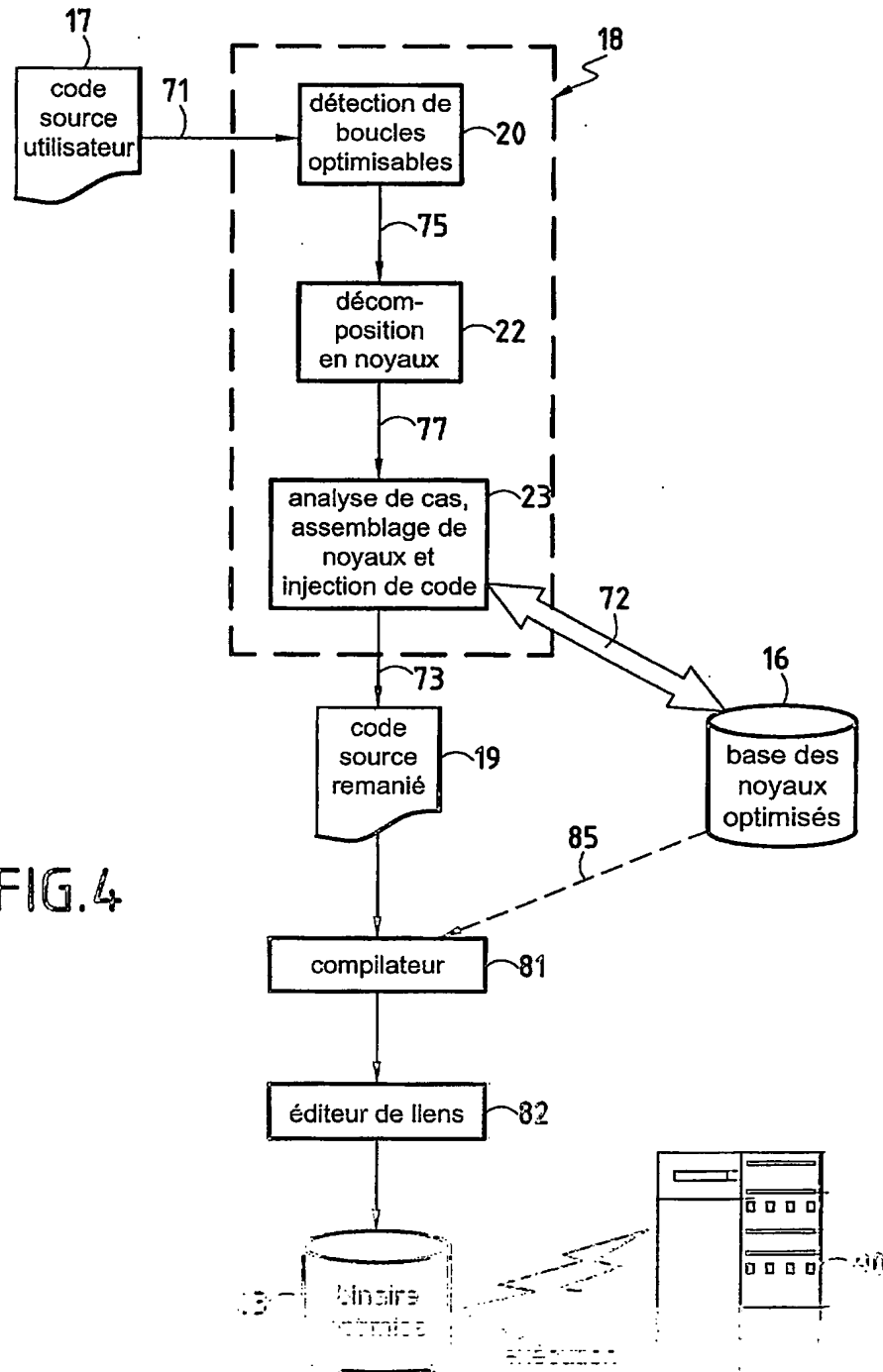
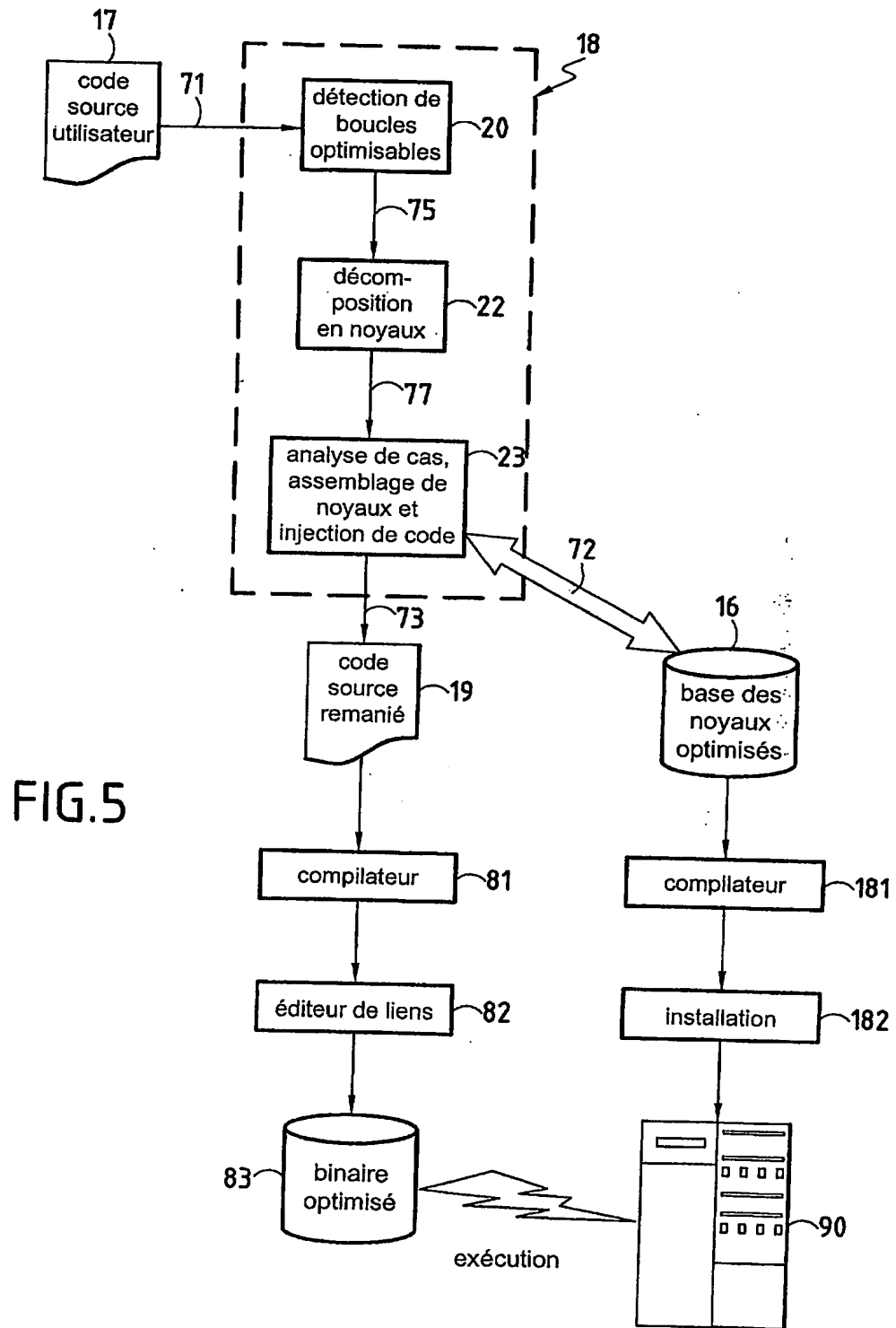


FIG.2



FIG.3





reçue le 28/04/04



# BREVET D'INVENTION

## CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI



### DÉPARTEMENT DES BREVETS

26 bis, rue de Saint Pétersbourg

75800 Paris Cedex 08

Téléphone : 01 53 04 53 04 Télécopie : 01 42 93 59 30

DÉSIGNATION D'INVENTEUR(S) Page N° 1.. / 3..

(Si le demandeur n'est pas l'inventeur ou l'unique inventeur)

Cet imprimé est à remplir lisiblement à l'encre noire

08 113 W / 260899

<b>Vos références pour ce dossier</b> (facultatif)		1H257420/18/JBT	
<b>N° D'ENREGISTREMENT NATIONAL</b>		04 00291	
<b>TITRE DE L'INVENTION</b> (200 caractères ou espaces maximum)			
Système de génération automatique de codes optimisés			
<b>LE(S) DEMANDEUR(S) :</b> COMMISSARIAT A L'ENERGIE ATOMIQUE			
<b>DESIGNE(NT) EN TANT QU'INVENTEUR(S) :</b> (Indiquez en haut à droite «Page N° 1/1» S'il y a plus de trois inventeurs, utilisez un formulaire identique et numérotez chaque page en indiquant le nombre total de pages).			
<b>Nom</b>		BODIN	
<b>Prénoms</b>		François	
<b>Adresse</b>	<b>Rue</b>	23, boulevard Charles Peguy	
	<b>Code postal et ville</b>	35700	RENNES
<b>Société d'appartenance (facultatif)</b>			
<b>Nom</b>		JALBY	
<b>Prénoms</b>		William	
<b>Adresse</b>	<b>Rue</b>	1, allée des Boeures	
	<b>Code postal et ville</b>	78124	MAREIL SUR MAULDRE
<b>Société d'appartenance (facultatif)</b>			
<b>Nom</b>		LE PASTEUR	
<b>Prénoms</b>		Xavier	
<b>Adresse</b>	<b>Rue</b>	20. rue Hoffmann	
	<b>Code postal et ville</b>	92340	BOURG LA REINE
<b>Société d'appartenance (facultatif)</b>			
<b>DATE ET SIGNATURE(S)</b> <b>DU (DES) DEMANDEUR(S)</b> <b>OU DU MANDATAIRE</b>			

reçue le 28/04/04



**BREVET D'INVENTION**  
**CERTIFICAT D'UTILITÉ**

Code de la propriété intellectuelle - Livre VI



**DÉPARTEMENT DES BREVETS**


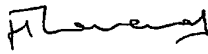
26 bis, rue de Saint Pétersbourg  
75800 Paris Cedex 08

Téléphone : 01 53 04 53 04 Télécopie : 01 42 93 59 30

**DÉSIGNATION D'INVENTEUR(S)** Page N° 2. / 3.  
(Si le demandeur n'est pas l'inventeur ou l'unique inventeur)

Cet imprimé est à remplir lisiblement à l'encre noire

DB 113 W / 260899

<b>Vos références pour ce dossier</b> (facultatif)		1H257420/18/JBT	
<b>N° D'ENREGISTREMENT NATIONAL</b>		04 00291	
<b>TITRE DE L'INVENTION</b> (200 caractères ou espaces maximum)  Système de génération automatique de codes optimisés			
<b>LE(S) DEMANDEUR(S) :</b> COMMISSARIAT A L'ENERGIE ATOMIQUE			
<b>DESIGNE(NT) EN TANT QU'INVENTEUR(S) :</b> (Indiquez en haut à droite «Page N° 1/1» S'il y a plus de trois inventeurs, utilisez un formulaire identique et numérotez chaque page en indiquant le nombre total de pages).			
<b>Nom</b>		LEMUET	
<b>Prénoms</b>		Christophe	
<b>Adresse</b>	<b>Rue</b>	124, Route des Charmes	
	<b>Code postal et ville</b>	78320	LEVIS-SAINT-NOM France
<b>Société d'appartenance (facultatif)</b>			
<b>Nom</b>		COURTOIS	
<b>Prénoms</b>		Eric	
<b>Adresse</b>	<b>Rue</b>	25, rue François-Charles Oberthur	
	<b>Code postal et ville</b>	35000	RENNES
<b>Société d'appartenance (facultatif)</b>			
<b>Nom</b>		PAPADOPOULO	
<b>Prénoms</b>		Jean	
<b>Adresse</b>	<b>Rue</b>	32, rue Dupressoir-Chailloux	
	<b>Code postal et ville</b>	92160	ANTONY
<b>Société d'appartenance (facultatif)</b>			
<b>DATE ET SIGNATURE(S)</b> <b>DU (DES) DEMANDEUR(S)</b> <b>OU DU MANDATAIRE</b> (Nom et qualité du signataire) Paris, le 28 avril 2004 THEVENET Jean-Bruno CPI 92-1236		<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <b>CABINET BEAU DE LOMENIE</b> 158, Rue de l'Université F 75340 PARIS CEDEX 07</div> 	

La loi n°78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés s'applique aux réponses faites à ce formulaire.  
Elle garantit un droit d'accès et de rectification pour les données vous concernant auprès de l'INPI.

reçue le 28/04/04



# BREVET D'INVENTION

## CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI



### DÉPARTEMENT DES BREVETS

26 bis, rue de Saint Pétersbourg  
75800 Paris Cedex 08

Téléphone : 01 53 04 53 04 Télécopie : 01 42 93 59 30

DÉSIGNATION D'INVENTEUR(S) Page N° 3. / 3..

(Si le demandeur n'est pas l'inventeur ou l'unique inventeur)

Cet imprimé est à remplir lisiblement à l'encre noire

DB 113 W / 260899

<b>Vos références pour ce dossier</b> (facultatif)		1H257420/18/JBT	
<b>N° D'ENREGISTREMENT NATIONAL</b>		04 00291	
<b>TITRE DE L'INVENTION</b> (200 caractères ou espaces maximum)			
Système de génération automatique de codes optimisés			
<b>LE(S) DEMANDEUR(S) :</b> COMMISSARIAT A L'ENERGIE ATOMIQUE			
<b>DESIGNE(NT) EN TANT QU'INVENTEUR(S) :</b> (Indiquez en haut à droite «Page N° 1/1» S'il y a plus de trois inventeurs, utilisez un formulaire identique et numérotez chaque page en indiquant le nombre total de pages).			
Nom		LECA	
Prénoms		Pierre	
Adresse	Rue	45, rue Laquintinie	
	Code postal et ville	75015	PARIS
Société d'appartenance (facultatif)			
Nom			
Prénoms			
Adresse	Rue		
	Code postal et ville		
Société d'appartenance (facultatif)			
Nom			
Prénoms			
Adresse	Rue		
	Code postal et ville		
Société d'appartenance (facultatif)			
Nom			
Prénoms			
Adresse	Rue		
	Code postal et ville		
Société d'appartenance (facultatif)			
<b>DATE ET SIGNATURE(S)</b> DU (DES) DEMANDEUR(S) OU DU COMMISSAIRE			